Comparison of CORE Network Emulation Platforms

Jeff Ahrenholz
Networked Systems Technology
Boeing Research & Technology
Seattle, WA

***Abstract*—CORE (Common Open Research Emulator) is a network emulation tool that uses virtualization provided by FreeBSD jails, Linux OpenVZ containers, and Linux namespaces containers. Here the modular architecture is described that makes supporting these three platforms possible, and the relative performance of each is compared. Initial work in integrating this tool with higher-fidelity link- and physical- layer emulation is also described.**

Keywords- network emulation, virtualization, routing, wireless, MANET

I. INTRODUCTION

The Common Open Research Emulator (CORE) is an open source network emulation framework. CORE uses existing operating system virtualization techniques to build wired and wireless virtual networks where protocols and applications can be run without modifying them. As a live emulator, these networks can be connected to real networks and systems, suitable for extending lab test-beds or testing with certain hardware-in-the-loop. The architecture consists of a GUI for easily drawing topologies, a services layer that instantiates lightweight virtual machines, and an API for tying them together. Topologies can also be scripted using the Python language. This modular architecture allows mixing and matching pieces. It has enabled CORE to be extended to work with FreeBSD jails, Linux OpenVZ containers, and Linux network namespace containers. Here these different platforms are compared in terms of virtualization features and performance.

CORE is typically used to emulate 10's of nodes on one machine. The focus is on emulating networking layers 3 and above for testing protocol and application software. Basic link effects such as bandwidth, delay, and error rate are offered. For more detailed link- and physical- layer modeling, CORE can be combined with other emulation and simulation tools such as EMANE [22] and ns-3 [4].

The remainder of this paper is organized as follows: related work is presented in Section 2. An overview of the CORE architecture is described in Section 3. Section 4 explains initial integration with EMANE for higher-fidelity link- and physical-layer emulation. Performance across the three different platforms is examined in Section 5. Section 6 describes the open source project, followed by conclusions and future work.

II. RELATED WORK

The Integrated Multiprotocol Network Emulator/Simulator (IMUNES) [1] is the predecessor to CORE and the basis for its Tcl/Tk GUI. It instantiated wired networks on FreeBSD 4.11 with a *vimage* (virtual image) patched kernel and the BSD Netgraph system. The author of IMUNES continued his work on network stack virtualization with the VirtNet project [2] for FreeBSD 7.x and 8.x, which is now part of the mainline FreeBSD kernel in the form of jails.

Simulation tools, such as ns-2 [3], ns-3 [4], OPNET [5], and QualNet [6], typically run on a single computer and abstract the operating system and protocols into a simulation model for producing statistical analysis of a network system. In contrast, network emulation tools, such as PlanetLab [7], NetBed [8], and MNE [9], often involve a dedicated testbed or connecting real systems under test to specialized hardware devices. CORE is a hybrid of the two types of tools, using virtualization to emulate the network stack of routers or hosts, and simulating the links that connect them together. This way it can provide the realism of running live applications on an emulated network while requiring relatively inexpensive hardware.

Machine virtualization tools, such as VirtualBox [10], VMware [11], Virtual PC [12], or Parallels [13], have become increasingly popular for running one or two instances of another operating system on standard hardware. Linux virtualization tools, such as Xen [14], UML [15], KVM [16], VServer [17], OpenVZ [18], and Linux Containers (LXC) [19], are mainly used for isolating multiple Linux server environments driven by the same hardware machine. Some of these techniques can be considered para-virtualization, where only part of the operating system is made virtual. This is the type of virtualization the CORE manages, using FreeBSD 8.x jails, Linux OpenVZ containers, or Linux network namespace containers for virtualization. In each of these virtualization platforms, processes are isolated in their own jail or namespace and associated with a virtual network stack. Machine hardware such as disks, video cards, timers, and other devices are not emulated, but shared between these nodes. Such lightweight virtualization allows CORE to scale to over a hundred virtual machines running on a single server.

From a network layering perspective, CORE provides high-fidelity emulation for the network layer and above, but uses a

simplified simulation of the link and physical layers. The operating system code implements the TCP/IP network stack, and user or system applications that run in real environments can run inside each emulated machine. The simplified links are realized using the Netgraph subsystem on FreeBSD and netem with bridging on Linux. Statistical network effects such as bandwidth, delay, and loss can be applied. For higher-fidelity link- and physical- layer emulation, CORE is being integrated with the ns-3 network simulator (using its real-time scheduler) and with the Extendable Mobile Ad-hoc Network Emulator (EMANE). Integration with EMANE is described in section 4.

III. CORE ARCHITECTURE

CORE aims at having a modular architecture that allows the mixing and matching of system components. That architecture is depicted in Fig. 1. The Tcl/Tk GUI provides a drawing canvas where nodes (routers, PCs, hubs, etc.) are placed and linked together in an editing mode. When a green "start" button is pressed, the GUI enters execute mode and the emulation is constructed. A screenshot of the GUI running in execute mode is shown in Fig. 2. The CORE services layer is responsible for instantiating the virtual nodes and networks. A custom, sockets-based API is used to communicate between the GUI and CORE services. Because the API operates over a TCP socket, the GUI does not need to run on the same machine as the emulation.

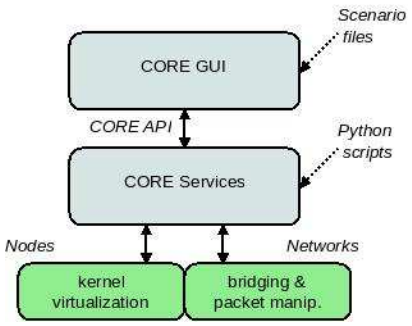

Figure 1. Overview of the CORE Architecture

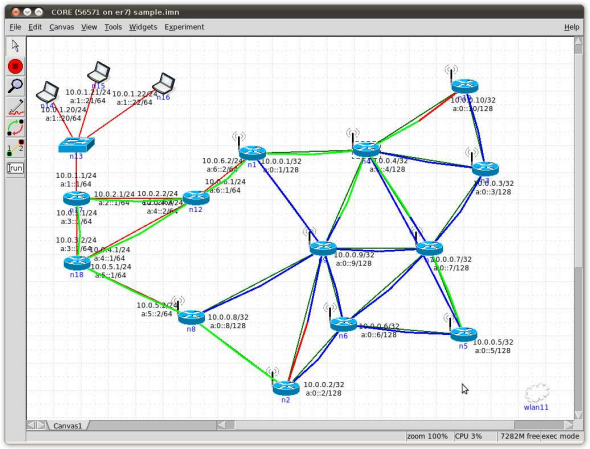

Figure 2. Screenshot of CORE GUI During Execution

During execution of the emulation, the GUI offers interactive shell access to each node, and customizable Widgets for displaying graphical node information. One of these widgets, for example, can display OSPF routing adjacencies between nodes using varying colored lines. This widget is also shown in Fig. 2. Services and applications are started on each node by configuring its startup script. Mobility scripts allow the user to script the movement of nodes after startup.

Each emulation consists of virtual nodes and virtual networks. Virtual Ethernet devices are created and installed into nodes and connected to the networks; they form the boundary between the two. Real network interfaces may also be installed into nodes, linking the virtual world with the real world. How the virtual nodes are implemented depends on the virtualization platform. The platform also determines which types of virtual networks will be available, see Table 1.

Starting with 8.0-RELEASE, the FreeBSD operating system has built-in support for jails with virtual network stacks; this support is available without patching the kernel source, but the option must be turned on and the kernel recompiled. The in-kernel Netgraph subsystem is unique to FreeBSD. This modular system allows arbitrarily linking together networking components. One of the Netgraph node types implements a network switch in CORE, for example, while another is used for setting up wireless LANs. The Netgraph system is lightweight and efficient, passing around packets by reference where possible.

TABLE I. VIRTUALIZATION POSSIBILITIES IN DIFFERENT VERSIONS OF CORE

CORE Version	Virtual Nodes	Virtual Networks
IMUNES, CORE 3.2-3.3	FreeBSD 4.11 vimages	Netgraph
CORE 3.4-3.5	FreeBSD 7.3 virtnet	Netgraph
CORE 3.5+	FreeBSD 8.0 jails	Netgraph
CORE 3.5+	Linux OpenVZ containers	netem + bridging
CORE 4.0+	Linux network namespace containers	netem + bridging
CORE 4.0+	Linux network namespace containers	EMANE
CORE 4.0+	Linux network namespace containers	ns-3

The next platform that CORE was extended to work with was Linux OpenVZ containers. The CORE daemon component of the CORE services layer is used for controlling OpenVZ containers. This container technology offers fairly lightweight virtual nodes along with various security controls. Each container typically has its own private filesystem which CORE circumvents by symlinking to one private core-root area. This way an application only needs to be installed once to be available in all containers. CORE also disables other controls such as the disk quota system and capability restrictions. OpenVZ is only available as a patch to the Linux kernel, so a separate 2.6.18 kernel (for the stable release) must be installed.

Namespaces support appeared in the mainline Linux kernel, starting with version 2.6.24, with the clone() system call. A namespace can have its own network stack, filesystem mounts, and process hierarchy, similar to the BSD jails. CORE was

extended to support these Linux network namespaces (or Linux containers) using a Python framework. The process that performs the clone() call becomes PID 1 in the new namespace, and new processes must be forked from that process. CORE uses its own vnoded daemon for this parent process that listens on a control socket for commands. A cored.py Python daemon runs at the CORE services layer to interact with the GUI using the CORE API. The Linux network namespace containers are similar to OpenVZ containers, but a little more lightweight without all of the access controls, and part of the mainline kernel. More recent Linux distributions such as Fedora 13 and Ubuntu 10.04 have namespaces support built-in, and do not require a special patched kernel.

Both Linux platforms use the *veth* virtual Ethernet pair driver, where one end of the virtual Ethernet pair lives inside the container and the other exists on the host. The CORE services bridges together veths on the host side using Linux Ethernet bridging. For forming wireless networks, Ethernet bridging tables (ebtables) is used to govern on/off connectivity between the devices on the bridge.

IV. EMANE INTEGRATION

The Linux Ethernet bridging and FreeBSD Netgraph system described in the previous section offer basic support for virtual networks. Both allow for statistical bandwidth limits, propagation delay, and packet errors, and an on/off connectivity for emulating wireless networks. A special WLAN Netgraph kernel module allows for per-node-pair effects, but that is limited to FreeBSD. For more realistic physical and link layer modeling, the virtual nodes created by CORE can be interconnected using other emulation and simulation tools, such as EMANE.

CORE has been integrated with the open source EMANE emulation system developed by CenGen Inc. in conjunction with the Naval Research Laboratory (NRL). EMANE is the successor to NRL's MANE and MNE tools, that provides pluggable MAC and PHY models tied together with an over-the-air (OTA) manager and event system. In the EMANE architecture, each Network Emulation Module (NEM) is composed of a MAC and PHY model that is tied to a Transport. The Transport glues the NEM together with a network interface. NEMs can connect with real or virtual interfaces. These components are shown in Fig. 3.

A. Instantiating Emulated Nodes and Networks

CORE provides lightweight virtual machines with virtual interfaces, and a detailed emulation of layers 3 and above, while EMANE provides detailed emulation for layers 2 and below. The two tools fit together nicely using the TUN/TAP virtual interface. A TAP virtual device can be pushed into a network namespace, while the socket end of the TAP is connected to the EMANE NEM using the Virtual Transport. Fig. 3 illustrates this boundary, where EMANE components are shown in a different shade than CORE the CORE components. The figure depicts all components on a single machine (or "Platform NEM Server"), but the virtual nodes and NEMs may also be distributed across several servers as described in the section after next.

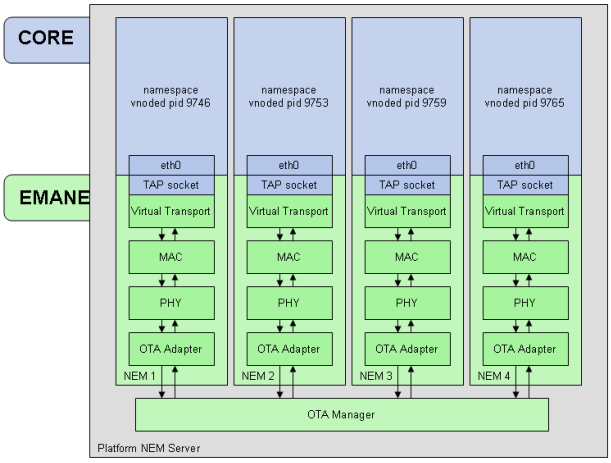

Figure 3. CORE/EMANE Integrated Architecture

The CORE GUI dynamically queries the CORE services for available models and parameters. The GUI configures model parameters using a wireless LAN node, because EMANE models radio interfaces. That configuration is then sent to the CORE services when the user presses the start button. The Python daemon in the CORE services layer automatically generates the required XML configuration files for the EMANE daemons, which are then launched by the Python program. The EMANE Virtual Transport must open each TAP device before it can be pushed into a network namespace.

B. Interacting with the Running Emulation

Once the emulation is running with CORE virtual nodes tied together with EMANE NEMs, there are two location spaces that need to be married. When a user moves a node on the CORE GUI canvas, an EMANE location event is generated by the CORE services and sent to all NEMs. Conversely, if an EMANE event generator, such as the MITRE mobility generator, publishes a location change event for a NEM, the CORE services will send that event to the CORE GUI and cause the node to move. The CORE GUI natively uses an {X, Y} Cartesian coordinate system while EMANE uses latitude, longitude, and altitude for location; the GUI has been extended to configure a scale and reference point for coordinate conversion.

C. Distributed Operation

The flexible EMANE architecture naturally supports distributed emulation. Using the CORE GUI, a selection of virtual nodes can be assigned to an emulation server. A CORE API message broker in the CORE services layer forwards messages from the GUI to the appropriate server. Each server may host several virtual nodes representing a portion of the overall emulation. The user needs to enable the over-the-air (OTA) manager for the EMANE model and configure it to use a physical interface on the system. When one virtual node sends a packet through its virtual TAP interface, EMANE will multicast that packet out the OTA interface so that all

emulation servers will receive the packet. The transport daemon bound to each individual virtual node then makes a decision on packet reception.

Distributing the emulation across several physical machines is one way to alleviate the processing bottleneck encountered on a single machine, especially as more complex and CPU-intensive models are used. Other performance considerations then arise, such as time synchronization between machines and the performance of the data network. Further study needs to be done to quantify the scalability of this distributed approach.

V. PERFORMANCE

CORE supports three different virtualization platforms for historical reasons. The GUI stems from the FreeBSD-based IMUNES project. Linux OpenVZ virtualization has been around for many years and was adopted by CORE due to its similarities with the FreeBSD jails. Linux network namespaces is a newer technology that is viewed as the successor to OpenVZ, having the advantage of being more lightweight at integrated with the mainline kernel. Having these three different virtualization platforms, it is desirable to understand the performance characteristics of each. Different CORE topologies are compared here for the FreeBSD 8.0, Linux OpenVZ 2.6.18, and Linux Netns 2.6.31 systems.

The hardware used for the performance tests shown here was an IBM x3550 rackmount server (machine type 7978), having a quad-core Xeon E5335 2.0GHz processor with 4MB cache, and configured with 2.0GB RAM, a 250GB 7200RPM SATA hard disk, and two onboard gigabit Ethernet ports. One of the two Ethernet interfaces was used to remotely run CORE over SSH with X11 forwarding. The other Ethernet interface was used as an RJ45 node in CORE.

The *iperf* utility is used to measure end-to-end TCP throughput along a network path. Here we convert the throughput reported by iperf to the *total system throughput*, by converting the megabits per second to packets per second, and then multiplying that by the number of hops traversed in the emulation. For example, if we are using a packet size of 50 bytes and get an iperf measurement of 10Mbps, we assume each hop sees (10,000,000 bits per second / (50 bytes/pkt × 8 bits/byte)) = 25,000 packets per second. If this network had 15 hops, the total system throughput would be (25,000 × 15) = 375,000 packets per second.

The packet size is varied by setting the iperf MSS parameter between 50, 500, 100, and 1500 bytes. The iperf client is run on a separate machine, not the CORE machine under test, to reduce the effects of generating the test load on the CPU usage. That machine is connected to a CORE network using the RJ45 (external interface) node. The CORE network consists of a chain of routers, with the iperf server running on the last node. The routers are configured with static routes. The number of hops used is varied between 5, 10, 15, 30, and 45 hop networks. Ten iperf runs were performed for each size network and each packet size, with the average shown here. The CPU usage was measured using the *vmstat* utility, which reports the idle CPU usage every second, and the average usage during the entire test was taken.

Fig. 4 shows the total system throughput calculated from the iperf throughput. It is important to note that the actual measured throughput is decreasing as the number of hops is increasing. This plot shows that varying the packet size and number of hops does not greatly affect the total packets per second. Each system has a limit in the number of packets per second that can be delivered through an emulation. The average CPU usage is shown in Fig. 5. Here it is worth noting that the FreeBSD system is able to more fully utilize the four processor cores, while the Linux systems are stuck with under 40% CPU used. FreeBSD is able to achieve a higher total system throughput due to its kernel-based Netgraph system.

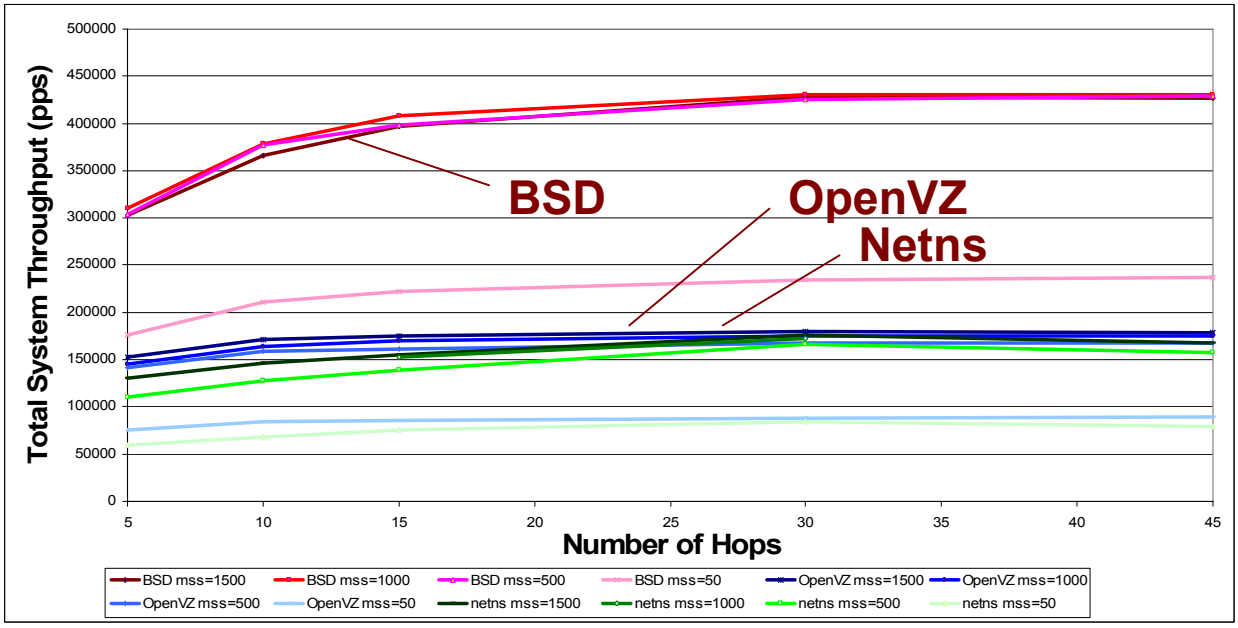

Figure 4. Total System Throughput derived from iperf tests

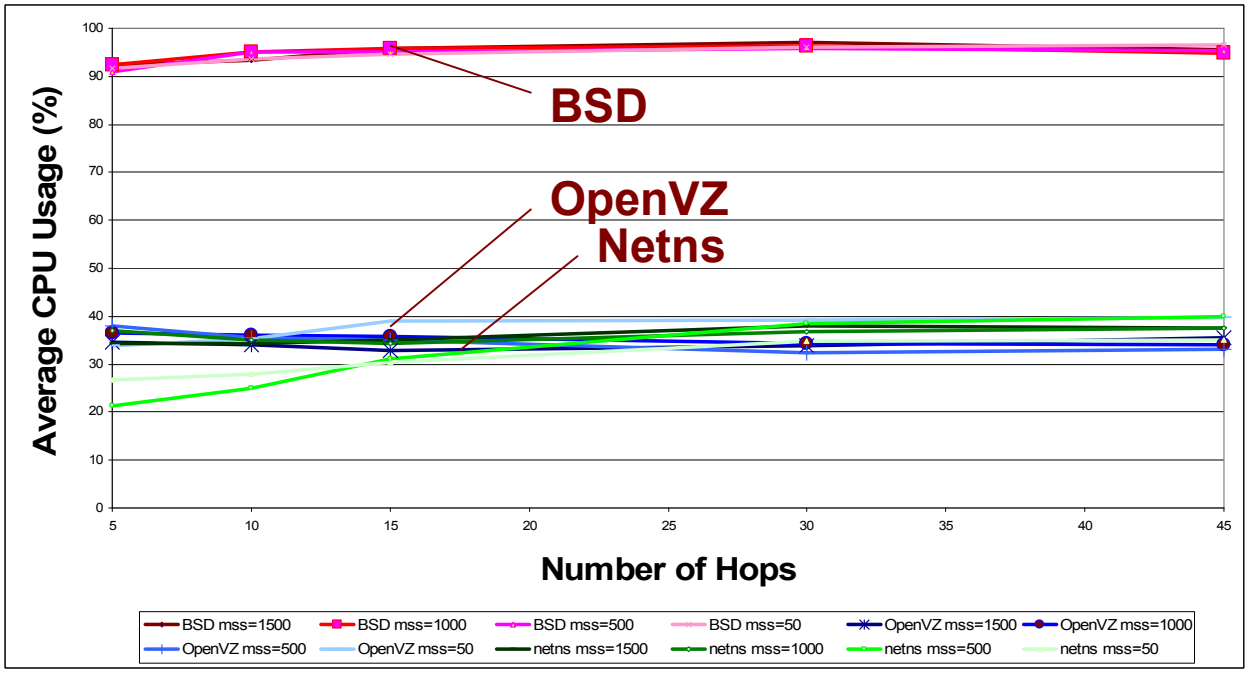

Figure 5. Average CPU Usage during Total System Throughput iperf tests

The second test performed was an iperf throughput test where both client and server are running on virtual nodes. This test shows the end-to-end throughput reported using the iperf TCP test and the CPU usage. The iperf utility is also run on the loopback device on the host and within a virtual machine on the system. Then the iperf measurements are taken between two nodes, and on 15 and 30 hop networks with static routers linked in a chain as before. Fig. 6 shows the throughput reported by iperf, while Fig. 7 shows the CPU usage. Again, FreeBSD is able to deliver greater throughput. Also the Linux network namespace containers are able to deliver slightly more throughput than the Linux OpenVZ containers while consuming fewer processor cycles.

To measure the latency through the system, the ping utility was used with a packet count of 1,000. The ping utility reports the average latency. Here the CORE networks were configured without any delay. This test was run on 15 and 30 hop networks with static routers linked in a chain. The average round-trip latency as reported by ping is shown in Fig. 8.

The final test shows real world usage of an emulated network, using the secure copy scp utility to transfer a large file across a network. An SSH server was run on the last router in the network. The scp client transfers a 1GB file over a chain of routers. The elapsed time in seconds is shown in Fig. 9, for the 15 and 30 hop networks. Lower times indicate better performance, with a higher throughput achieved. The average CPU usage is shown in Fig. 10. The FreeBSD system again achieved higher throughput and was able to utilize more of the CPU resources.

VI. Open Source Project

CORE is free software licensed under the 3-clause BSD license and hosted on the Naval Research Laboratory (NRL) website [20]. Public mailing lists provide a support and discussion area for users, and a Wiki [21] offers pages of tips and other information that change more rapidly than the website and user manual. Contributions back to the project are encouraged, including bug reporting or technical suggestions.

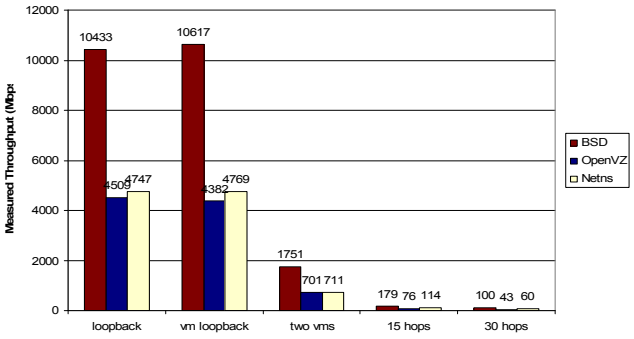

Figure 6. Iperf Measured Throughput on one machine

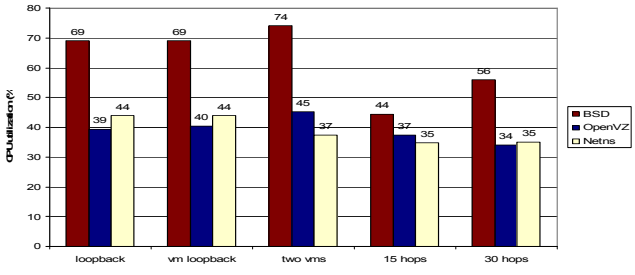

Figure 7. Average CPU Usage during iperf test on one machine

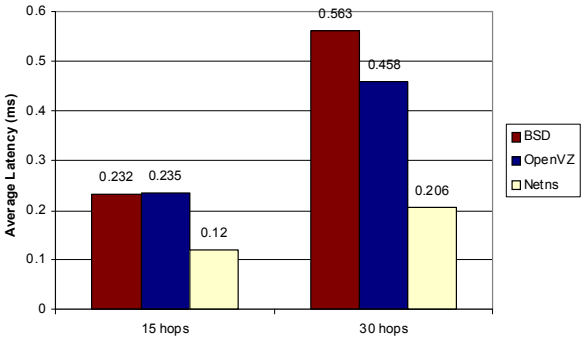

Figure 8. Average Round-trip Latency for ping test

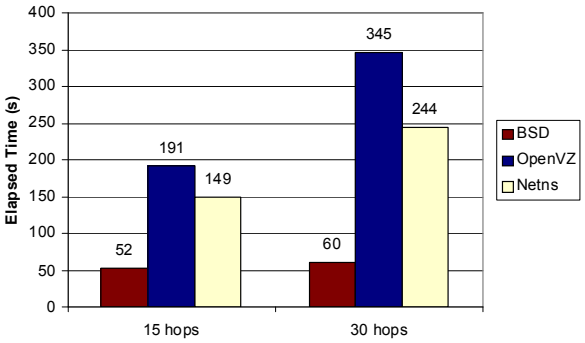

Figure 9. Elapsed Time for scp 1GB File Transfer

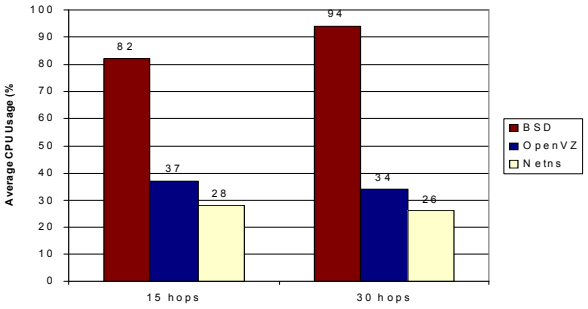

Figure 10. CPU Usage for scp 1GB File Transfer

VII. Conclusion and Future Directions

The architecture of CORE was described along with some related work. Integration with the EMANE emulation system was briefly covered to introduce the possibility of using higher-fidelity link and physical layer models. The performance of a few scenarios was compared for all three supported platforms, with iperf throughput tests, ping latency tests, and scp file transfer tests. FreeBSD, with its Netgraph subsystem utilizes more CPU to deliver more packets in all of the tests, while the Linux systems, both using Linux Ethernet bridging, perform similarly. Finally the open source project was summarized with links for more information.

Future work continues on CORE to migrate to a unified Python-based code base, which increases the ability to script emulations without using the GUI. Health monitoring systems and feedback to the user when encountering system resource limits are under development. Distributed emulation across multiple physical emulation servers helps address scalability limitations, but was not addressed in this paper. Performance of the combined CORE with EMANE and ns-3 link/physical layers should be investigated. Live migration of virtual machines from one server to another is supported in both the Linux virtualization technologies and would make an interesting feature addition to CORE. Finally, since CORE is concerned with instantiating and controlling virtual machines it would be worthwhile to generalize CORE for the control of live test-beds.

Acknowledgments

This work was done in cooperation with the NRL Protean Research Group, Code 5522 under the OSD-sponsored Network Communications Capability Program (NCCP).

References

[1] Zec, M., and Mikuc, M. “Operating System Support for Integrated Network Emulation in IMUNES”, ACM ASPLOS XI, October 2004.

[2] Zec, M. “Network Stack Virtualization”, EuroBSDCon 2007, September 2007.

[3] “The Network Simulator - ns-2”, http://www.isi.edu/nsnam/ns/

[4] “The ns-3 Network Simulator”, http://www.nsnam.org/

[5] “OPNET Modeler: Scalable Network Simulation”, http://www.opnet.com/solutions/network_rd/modeler.html

[6] “Scalable Network Technologies: QualNet Developer”, http://www.scalable-networks.com/products/qualnet/

[7] “PlanetLab: an open platform for deploying…”, http://www.planet-lab.org/

[8] “Emulab - Network Emulation Testbed Home”, http://boss.netbed.icics.ubc.ca/

[9] “Mobile Network Emulator (MNE)”, http://cs.itd.nrl.navy.mil/work/proteantools/mne.php

[10] “VirtualBox”, http://www.virtualbox.org/

[11] “VMware Server”, http://www.vmware.com/products/server/

[12] “Microsoft Windows Virtual PC”, http://www.microsoft.com/windows/virtual-pc/

[13] “Parallels Desktop for Windows & Linux”, http://www.parallels.com/products/desktop/pd4wl/

[14] “Xen Hypervisor”, http://www.xen.org/products/xenhyp.html

[15] “The User-Mode Linux Kernel”, http://user-mode-linux.sourceforge.net/

[16] “Kernel Based Virtual Machine”, http://kvm.qumranet.com/kvmwiki

[17] “Linux-VServer”, http://linux-vserver.org/

[18] “OpenVZ Wiki”, http://wiki.openvz.org/Main_Page

[19] Helsley, M. “LXC: Linux container tools”, IBM devloperWorks Technical Library. Feb 2009.

[20] “Networks and Communication Systems Branch: CORE”, http://cs.itd.nrl.navy.mil/work/core/index.php

[21] “CORE wiki home”, http://code.google.com/p/coreemu/wiki/Home

[22] “Networks and Communication Systems Branch: EMANE”, http://cs.itd.nrl.navy.mil/work/emane/index.php